STAT 672 Final Project: Stochastic Gradient Descent

Tom Wallace

April 16, 2018

1 Introduction

1.1 Organization

This paper is divided into four sections. The remainder of this **Introduction** section gives intuitive motivation for stochastic gradient descent (SGD). The **Method and Theory** section more rigorously presents the mathematics of SGD and some of its notable properties. The **Applications** sections highlights the real-world settings and uses of SGD, including a case study data analysis. The **Conclusion** section summarizes overall findings.

1.2 Motivation

Optimization is fundamental to statistical modeling. The chief task of statistical modeling is to characterize the relationship between explanatory variables and an outcome variable, and the chief method for doing so is to estimate values for coefficients that best relate each explanatory variable to the outcome variable. The term "best" implies picking coefficient values that maximize some measure of goodness (e.g. likelihood) or minimize some measure of badness (e.g. loss function). Mathematical optimization is the typical route to achieving such minimization or maximization. Two important considerations for optimization are parametric assumptions and computational complexity. SGD, an optimization technique, is particularly motivated by these considerations.

1.2.1 Parametric vs. non-parametric

Assuming that the outcome variable follows a particular statistical distribution aids the computation of optimal coefficients. For example, assumptions in ordinary least squares (OLS) regression—assumptions that readers almost certainly are familiar with and so will not be repeated here—allow a closed form solution. The optimization problem of choosing coefficient $\hat{\boldsymbol{\beta}}$ that minimize squared error is solved by $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$ (where $\boldsymbol{X}$ is the feature matrix and $\boldsymbol{Y}$ the outcome variable).

Even if a parametric model does not have a closed-form solution, the parametric assumption allows some useful optimization techniques. Consider logistic regression. The maximum likelihood estimator (MLE) approach for estimating coefficients leads to a system of D equations. This system of equations typically is numerically solved using the iterative Newton-Raphson algorithm:

$$\hat{\boldsymbol{\beta}}_{n+1} = \hat{\boldsymbol{\beta}}_n - \boldsymbol{H}^{-1}(\hat{\boldsymbol{\beta}}_n)\boldsymbol{J}(\hat{\boldsymbol{\beta}}_n)$$

$\boldsymbol{J}$ is the Jacobian (the first derivative of the log-likelihood function l with respect to each w_j) and $\boldsymbol{H}$ is the Hessian (the second derivative of l with respect to $w_j, w_{j'}$). The practicality of Newton-Raphson thus depends on whether it is convenient to find $\boldsymbol{J}$ and $\boldsymbol{H}$. It is convenient for logistic regression because parametric and independent-and-identically-distributed (IID) assumptions mean l is a simple sum of the log probability distribution function (PDF, in this case binomial) for each observation. We "know" (assume) the form of this PDF and so are confident that the second derivative exists and is not too onerous to calculate. In non-parametric settings, we often cannot be so certain and face the possibility of $\boldsymbol{H}$ being non-existent or cumbersome.

The need to conduct optimization in non-parametric settings is a chief motivation for gradient descent (GD), of which SGD is a variant. In non-parametric settings—most notably supervised and unsupervised statistical learning, in which we again seek to find optimal coefficients to relate input variables to output variables for the purposes of classification or regression—there typically is no closed form solution for the coefficients. It also may not be convenient to find and evaluate the Hessian, making Newton-Raphson undesirable. SGD does not require any parametric assumptions. In its most basic form, SGD only requires finding the gradient (though some extensions do need the Hessian or an approximation to it). SGD thus is well-suited for non-parametric settings.

1.2.2 Computational Complexity

How an optimization technique scales with sample size n is another important consideration. It is little comfort if a method reaches the correct solution but requires an excessive amount of time to do so. "Plain" or "batch" GD requires evaluating the gradient for every single observation, every single iteration, until the algorithm converges. For example, for a dataset of $n = 10^6$ that required 25 iterations to converge, batch GD would require evaluating the gradient 25×10^6 times. This scaling with n can cause untenably long computation time.

SGD alleviates these computational difficulties by requiring the gradient to be evaluated for only a single randomly chosen observation per iteration. This approach means convergence is "noisier" and hence requires more iterations to converge, but each iteration is less complex to compute and so can be done faster. SGD thus scales much more favorably with n than GD, and so is particularly useful for large-n applications such as machine learning and big data problems.

2 Method and Theory

2.1 Basic Form

This section presents the basic form of GD and SGD to set up a more detailed examination in subsequent sections. Suppose we want to minimize a function $f : \mathbb{R}^D \to \mathbb{R}$.

$$\min_w f(w) \tag{1}$$

f takes as input $w \in \mathbb{R}^D$. Assume that f is convex and differentiable, i.e. we can compute its gradient with respect to w, $\nabla f(w)$. The optimal w, i.e. that which minimizes (1), is denoted $\hat{w}$. The iterative GD algorithm for finding $\hat{w}$ is:

$$w^{(t+1)} = w^{(t)} - \gamma \nabla f(w^{(t)}) \tag{2}$$

t refers to a particular iteration. Assume that we have supplied an initial starting guess for w for $t = 0$, $w^{(0)}$. γ refers to step size (also called learning rate). Assume for now that γ is fixed. The GD algorithm iterates until some stop condition is met. This may be a fixed number of iterations, or that the quality of approximation meets some predefined threshold. A common stopping condition is when the L2 norm of the gradient is less than some arbitrarily small constant.

$$||\nabla f(w^{(t)})||_2 \leq \epsilon \tag{3}$$

Consider a modified situation. Suppose f now takes two arguments, w and $X \in \mathbb{R}^D$. $f : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. We have n observations of X, and denote by $\boldsymbol{X}_{n \times D}$ the matrix of these observations, with X_i being a particular observation or row in that matrix. We apply f over all observations, i.e. $\frac{1}{n} \sum_{i=1}^n f(w, X_i)$. Our new problem is:

$$\min_w \frac{1}{n} \sum_{i=1}^n f(w, X_i) \tag{4}$$

We could apply the GD algorithm above to find the optimal w.

$$w^{(t+1)} = w^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^n \nabla f(w^{(t)}, X_i) \tag{5}$$

But, note that doing so requires evaluating the gradient at every single observation $i \leq n$. This may be computationally intractable for large n and high-dimensional D. The innovation of SGD is to instead evaluate only a single randomly-chosen i at each iteration t.

$$w^{(t+1)} = w^{(t)} - \gamma \nabla f(w^{(t)}, X_i) \tag{6}$$

As it turns out, SGD converges to $\hat{w}$ and does so in a computationally advantageous way compared to GD.

2.2 Key Properties

2.2.1 Correctness

This section gives both intuition and (partial) proof for why (S)GD converges to $\hat{w}$.

Intuitively, our assumption that f is convex means that there is one critical point and it is the global minimum. Basic calculus tells us that this critical point is located where $f' = 0$ (in one dimension) or $\nabla f = 0$ (in higher dimensions). So our task is to search in the space of f for that point. We start with some initial guess $w^{(0)}$, and every iteration, move "down" the gradient in search of zero. Every iteration, we check if the gradient is equal to 0; if not, we keep moving "down." If the gradient is arbitrarily close to zero, then we have found the critical point, which must be the value of w that minimizes f and hence is $\hat{w}$ (or at least a close approximation to it).

For a more formal proof—following that of Tibshirani 2013—assume (in addition to previously stated assumptions that f is convex and differentiable) that the gradient of f is Lipschitz-continuous with constant L, i.e. that $||\nabla f(x) - \nabla f(y)||_2 \leq L||x - y||_2$ for any x, y. This implies that $\nabla^2 f(w) - LI$ is a negative semi-definite matrix. We perform a quadratic expansion of f around $f(x)$ and obtain the following inequality:

$$\begin{aligned} f(y) &\leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}\nabla^2 f(x)||y - x||_2^2 \\ &\leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}L||y - x||_2^2 \end{aligned} \tag{7}$$

Now, suppose we use the GD algorithm presented in (2) with $\gamma \leq \frac{1}{L}$. Denote $x^+ = x - \gamma\nabla f(x)$ and substitute x^+ in for y:

$$\begin{aligned} f(x^+) &\leq f(x) + \nabla f(x)^T(x^+ - x) + \frac{1}{2}L||x^+ - x||_2^2 \\ &= f(x) + \nabla f(x)^T(x - \gamma\nabla f(x) - x) + \frac{1}{2}L||x - \gamma\nabla f(x) - x||_2^2 \\ &= f(x) - \nabla f(x)^T\gamma\nabla f(x) + \frac{1}{2}L||\gamma\nabla f(x)||_2^2 \\ &= f(x) - \gamma||\nabla f(x)||_2^2 + \frac{1}{2}L\gamma^2||\nabla f(x)||_2^2 \\ &= f(x) - (1 - \frac{1}{2}L\gamma)\gamma||\nabla f(x)||_2^2 \end{aligned} \tag{8}$$

We have defined $\gamma \leq \frac{1}{L}$. This implies:

$$-(1 - \frac{1}{2}L\gamma) = \frac{1}{2}L\gamma - 1 \leq \frac{1}{2}L\frac{1}{L} - 1 = \frac{1}{2} - 1 = -\frac{1}{2}$$

Returning to (8), we obtain:

$$f(x^+) \leq f(x) - \frac{1}{2}\gamma||\nabla f(x)||_2^2 \tag{9}$$

By definition, a squared L2 norm will always be positive unless its content is 0, and we have defined γ to be positive, so $\frac{1}{2}\gamma||\nabla f(x)||_2^2$ will always be positive unless the gradient is equal to zero. So, (9) implies that GD results in a strictly decreasing objective function value until it reaches the point where the gradient equals zero, which is the optimal value. A key caveat is an appropriately chosen γ, a point covered in more detail later.

The above proof is for GD. A rigorous proof is not given for why SGD also converges to the optimal value. Informally, we can note that the above proof says given infinite t and appropriate γ, the iterative algorithm will always converge to the optimal value. SGD is doing the same thing as GD, just with a single observation per iteration rather than the entire dataset per iteration. Since SGD is using less information per iteration, we would expect the convergence to require more iterations. But per (9) we expect it to *eventually* arrive at the optimal solution. If so, the difference between GD and SGD must then primarily revolve around convergence speed, not the final value that is converged to.

2.2.2 Speed

Table 1—a simplified version of that in Bottou 2010—illustrates the computational advantages of SGD over GD. As shown in row 1, because SGD uses less information per iteration than GD, it requires more iterations to achieve some fixed degree of accuracy ρ. However, row 2 shows that because GD computes the gradient for every observation every iteration, while SGD only does so for a single randomly chosen iteration, GD's time per iteration scales linearly with n while SGD's is a constant. Thus, we conclude that SGD's time to reach accuracy ρ scales only with ρ, while GD's scales with both ρ and—crucially—linearly with n. So, the larger n grows, the larger SGD's computational advantage over GD.

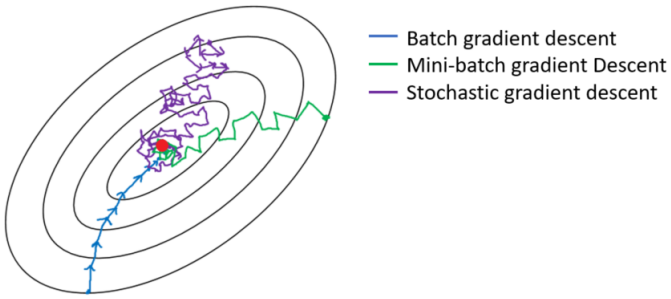
Table 1: Asymptotic comparison of GD and SGD

	GD	**SGD**
Iterations to accuracy ρ	$\log \frac{1}{\rho}$	$\frac{1}{\rho}$
Time per iteration	n	1
Time to accuracy ρ	$n \log \frac{1}{p}$	$\frac{1}{\rho}$

Figure 1 visually compares SGD's convergence to that of GD (and mini-batch gradient descent, a variation addressed later in this paper).[1] The path of SGD's convergence is very squiggly, reflecting the extra variation introduced by only using a single observation per iteration. The path of GD's convergence is much smoother, reflecting the use of all observations per iteration.

[1] Image taken from www.towardsdatascience.com

Figure 1: Convergence



2.3 Extensions

The basic SGD algorithm has been extended in many different ways. The popularity of the algorithm disallows a comprehensive or detailed treatment of all developments. This sub-section covers some of the more important and interesting ones.

2.3.1 Step Size

As hinted at in our proof of convergence to $\hat{w}$, much depends on an appropriately chosen step size γ. If γ is too small, gradient descent may take an excessively long time to run (since we are only moving a small distance "down" the gradient every iteration); if γ is too large, then an iteration's step may "overshoot" the critical point. An entire literature has developed around the best way to select γ. The central insight that is common to most γ-selection methods is to treat it as a time-indexed rather than fixed hyperparameter, i.e., $\gamma^{(t)}$ rather than γ. Ideally we would like $\gamma^{(t)}$ to be large when far away from the critical point (so as to increase speed of convergence) and to be small when close to the critical point (so as to avoid overshooting).

Line search is one method for computing step size. As described in Boyd and Vandenberghe 2004, there are two main variants: *exact* and *backtracking*. In exact line search, $\gamma^{(t)}$ is chosen to minimize f along the ray $\{x + \gamma^{(t)}\Delta x\}$, where $\gamma^{(t)} \in \mathbb{R}^+$) and Δx is the descent direction determined by SGD:

$$\gamma^{(t)} = \underset{s \geq 0}{\operatorname{argmin}}\, f(x + s\Delta x) \tag{10}$$

This obviously is an optimization problem in and of itself, and so it can be computationally impractical to add this burden in addition to the "main" optimization problem we are trying to solve using SGD. For this reason, *backtracking* is an iterative approximation method that is computationally lighter. The backtracking algorithm takes two hyper-parameters $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$. It starts with an initial guess of $s^{(0)} = 1$, and then for every iteration t, updates

$s := \beta t$. The algorithm stops when $f(x + s\Delta x) \leq f(x) + \alpha s \Delta f(x)^T \Delta x$, and the final value of s is taken as $\gamma^{(t)}$.

Bottou 2012 advocates using learning rates of the form $\gamma^{(t)} = \gamma^{(0)}(1 + \gamma^{(0)}\lambda t)^{-1}$. When the Hessian H of f is strictly positive, using $\gamma^{(t)} = (\lambda_{\min} t)^{-1}$, where $\lambda_{\min}$ is the smallest eigenvalue of H, produces the best convergence speed. Note that $(\lambda_{\min} t)^{-1}$ decreases asymptotically with t, matching our intuition about larger step sizes in early iterations and smaller step sizes in later iterations. However, simply using $\gamma^{(t)} = (\lambda_{\min} t)^{-1}$ can produce *too* large of steps in early iterations. Hence, it often works better to start with some reasonable initial estimate $\gamma^{(0)}$ that then decays in the fashion of $(\lambda_{\min} t)^{-1}$, leading to the expression in the first sentence of this paragraph. By definition, "reasonable" is more of a judgment call than a mathematical statement: Bottou provides some heuristics for creating such an estimate of $\gamma^{(0)}$.

2.3.2 Momentum and Acceleration

In our set-up of SGD, we stipulated that the function to be minimized, f, is convex. Although this is a necessary condition for some proofs of SGD's correctness, SGD is widely used in applications—most prominently, neural networks and deep learning—where this assumption is not always true. In such settings, it is possible that there are multiple local minima, and/or that there are areas where the surface curves much more steeply in one dimension than another. The challenge for SGD is how to avoid getting "stuck" in these local ravines and continue iterating until the true global minimum is achieved. Even if there is no pathological curvature, incorporating information about the local topology may result in more efficient algorithms and faster convergence.

Momentum—also called acceleration—is a common modification of GD. As outlined in Rumelhart, Hinton, McClelland, et al. 1986 and Qian 1999, we add a new acceleration term to the familiar GD equation. Let $z^{(t+1)} = \alpha z^{(t)} + \nabla f(w^{(t)})$. GD with momentum then is:

$$w^{(t+1)} = w^{(t)} - \gamma z^{(t+1)} \tag{11}$$

Now, the weight vector obtained for the current timestep depends both on the current gradient *and the update of the previous time-step*, with the acceleration parameter α governing how much weight is given to the previous time-step. This formulation leads to momentum: dimensions whose gradients point in the same direction have proportionally greater updates, while dimensions whose gradients exhibit strong curvature have proportionally lesser updates. The effect is to help stop (S)GD from oscillating in ravines and speed up convergence. There are many variations on acceleration, including Nesterov accelerated gradient (NAG) (Nesterov 1983), Adam (Kingma and Ba 2014), AdaGrad (Duchi, Hazan, and Singer 2011), and AdaDelta (Zeiler 2012).

2.3.3 Averaging

Averaged SGD (ASGD)—to the author's knowledge first proposed in Polyak and Juditsky 1992—is another variation. Typically, we consider $\hat{w}$ (the optimal solution) to be that associated with the final iteration of SGD, when some stop condition is met. For example, in a situation where we run N iterations before stopping, $\hat{w} = w^{(N)}$. In ASGD, we consider $\hat{w}$ the average of w across all iterations, i.e.

$$\hat{w} = \bar{w} = \frac{1}{N} \sum_{t=1}^{N} w^{(t)} \tag{12}$$

The value proposition of ASGD is that sometimes SGD will oscillate around the optimal solution. By taking the average across all updates, ASGD reduces this noise and is likely to give a solution closer to the optimum. More sophisticated versions of ASGD have been proposed in Zhang 2004 and Xu 2011.

2.3.4 Mini-batch

Mini-batch gradient descent (MGD) can be regarded as a mid-point between batch and stochastic gradient descent in that it uses more than one but less than all of the available observations. Suppose we have n observations and divide them into m groups called mini-batches, each of equal size $b = \frac{n}{m}$. Now, at every iteration t, randomly pick one of the mini-batches (all are equally likely to be chosen). For that mini-batch:

$$w^{(t+1)} = w^{(t)} - \gamma \frac{1}{b} \sum_{i=1}^{b} \nabla f(w^{(t)}) \tag{13}$$

MGD iterates with t and converges to $\hat{w}$ in the normal manner. Every update, a new b is randomly chosen and the above algorithm iterated. Dekel et al. 2012 and Li et al. 2014 analyze MGD's speed of convergence and suggest that has some advantageous qualities for parallelization, a topic that is explained in more depth in the following subsection.

2.3.5 Parallelization

SGD is commonly used in large-n applications. Thus, even though SGD is a computational improvement over batch GD, there has been interest in whether SGD can be made even faster by parallelizing it. As background, typical computer applications are serial: problems are broken down into a discrete series of instructions that are executed one after another on a single processor. Parallel computing calls for breaking down problems into discrete parts that can be solved concurrently and are executed simultaneously on multiple processors (Barney et al. 2010). There are many paradigms of parallel computing but a useful high-level distinction is between shared memory, in which all processors share a common address space, and distributed memory, in which each processor or group of processors has its own address space. The latter has become

increasingly popular in industry via Apache Hadoop and associated projects (e.g. Spark). Distributed memory applications often require very complex computing strategies for how to combine the work of many processors without a shared memory space into a single coherent answer.

This background helps to emphasize the novelty of the parallel implementation of SGD proposed in Zinkevich et al. 2010. Though backed by highly technical proofs, the actual algorithm is strikingly simple. Suppose we have N processors. We seek to conduct SGD to estimate w across n observations with fixed learning rate γ for some fixed number of steps T.

Algorithm 1: Parallel SGD

Define $T = \frac{n}{N}$;
Randomly partition examples, giving T to each machine;
forall $i \in \{1 \dots N\}$ **do**
    Randomly shuffle data on machine i;
    Initialize $w^{i,(0)} = 0$;
    forall $t \in \{1 \dots T\}$ **do**
        $w^{i,(t+1)} = w^{i,(t)} - \gamma \nabla f(w^{i,(t)})$;
    end
end
Aggregate from all computers: $\bar{w} = \frac{1}{N} \sum_{i=1}^{N} w^{i,(T)}$;
return $\bar{w}$

Zinkevich et al. show that this algorithm has a number of desirable properties. It requires no communication between machines until the end; asymptotically, the error approaches zero; and, the amount of time required is independent of the number of examples (though this is an inherent property of SGD and is not unique to their algorithm). Further improvements have been proposed: e.g., the Hogwild algorithm presented in Recht et al. 2011 deals with locking issues in parallel computation of SGD.

3 Applications

In theory, SGD is agnostic to f and w as long as they meet required assumptions, and so can be used in many different areas of optimization. In practice, SGD is overwhelmingly popular in the statistical learning field for estimating weight coefficients in predictive models.

3.1 SGD and Statistical Learning

Consider a typical supervised classification problem. We have feature matrix $\boldsymbol{X}_{n \times D}$ (corresponding to n observations and D features) and labels $\boldsymbol{Y}_{n \times 1}, Y_i \in \{-1, 1\}$. The goal is to predict a particular observation's label Y_i using that observation's features $\boldsymbol{X}_i$. To emphasize the utility of SGD, suppose that both n and D are very large. We have a hypothesis class $\mathcal{F}$ consisting of various

functions $f_{\boldsymbol{w}}(\boldsymbol{X}_i) \in \mathcal{F}$ parametrized by weight vector $\boldsymbol{w}$. We have convex loss function $L(Y_i, f_{\boldsymbol{w}}(\boldsymbol{X}_i))$ that expresses the cost of misclassification. We will consider the optimal function $\hat{f}_{\boldsymbol{w}}(\boldsymbol{X}_i) \in F$ that which minimizes empirical risk over all observations: $\frac{1}{n}\sum_{i=1}^{n} L(Y_i, f_{\boldsymbol{w}}(\boldsymbol{X}_i))$. Denote $\hat{\boldsymbol{w}}$ the weight coefficients of this optimal function. We thus have:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(Y_i, f_{\boldsymbol{w}}(\boldsymbol{X}_i)) \tag{14}$$

How shall we solve this optimization problem? Newton's method requires calculating the Hessian, which will have D^2 entries; since D is large, this is inconvenient to compute. We could try batch gradient descent:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{w}} L(Y_i, f_{\boldsymbol{w}^{(t)}}(\boldsymbol{X}_i)) \tag{15}$$

But, in line with our analysis in section 2.2.2, batch GD requires evaluating the gradient over every observation $i \leq n$ for every iteration t. Since we have defined n to be large, this is computationally inconvenient. Rather, we use SGD to evaluate only a single randomly chosen observation per iteration.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \nabla_{\boldsymbol{w}} L(Y_i, f_{\boldsymbol{w}^{(t)}}(\boldsymbol{X}_i)) \tag{16}$$

Per previous analyses we know that this approach will converge to an arbitrarily good answer (if certain assumptions are met); will converge to that arbitrary degree of precision in less time than plain gradient descent; does not necessarily require computing the Hessian; and can be parallelized across many machines. These attractive qualities make SGD well-represented in almost all areas of machine learning. In the following subsection, we take a closer look at one particularly prominent applications, support vector machines.

3.1.1 Support vector machines

Originally proposed by Boser, Guyon, and Vapnik 1992 and Cortes and Vapnik 1995, support vector machines (SVM) are among the best supervised learning algorithms. Here we set up the SVM problem and then show how SGD can be used to solve it.

Shalev-Shwartz et al. 2011

3.2 Case Study

Dal Pozzolo et al. 2015

3.3 Real-World Applications

If a Silicon Valley press release uses words like "artificial intelligence", "machine learning", and "big data" to describe a new product or service, it very likely

uses SGD in some way. Such is the extent of SGD's popularity. Below is a list of particularly prominent examples of SGD's practical applications.

- **ImageNet** is a dataset consisting of millions of high-resolution, labeled images. The ImageNet Large Scale Visual Recognition Competition (ILSVRC) is an annual contest in which competing teams train models for automatic image classification. The submission of Krizhevsky, Sutskever, and Hinton 2012 for the 2012 ILSVRC used convolutional neural nets to achieve dramatic reduction in error rates and is one of the most cited machine learning papers of all time. Per the aforementioned paper: "We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.00005."
- Google's **AlphaGo** software. AlphoGo applies deep learning to the board game of Go and has received extensive attention for its ability to defeat even top human players (e.g., twice making the cover of *Nature* with Silver, Huang, et al. 2016 and Silver, Schrittwieser, et al. 2017). Google's slides from the 2016 International Conference on Machine Learning (ICML) indicate that SGD is used in every stage of the model: supervised learning of policy networks, reinforcement learning of policy networks, and reinforcement learning of value networks.
- The **Netflix Grand Prize** was a competition hosted by Netflix in which teams competed to develop models for using Netflix data to make better movie recommendations (Bennett, Lanning, et al. 2007). Most of the top entries used SGD for estimating model coefficients; for example, Koren 2009 states that "in order to learn the involved parameters... learning is done by a stochastic gradient descent algorithm running for 30 iterations."
- Google's **word2vec** natural language processing (NLP) software powers Google Translate, among other applications. Mikolov et al. 2013 and Rong 2014 discuss the continuous bag of words (CBOW) model underlying word2vec and how SGD is used to estimate coefficients in the model.

4 Conclusion

References

[1] Blaise Barney et al. *Introduction to parallel computing*. Lawrence Livermore National Laboratory, 2010.

[2] James Bennett, Stan Lanning, et al. "The netflix prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA. 2007, p. 35.

[3] Bernhard E Boser, Isabelle M Guyon, and Vladimir Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152.

[4] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[5] Léon Bottou. "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[8] Andrea Dal Pozzolo et al. "Calibrating probability with undersampling for unbalanced classification". In: *Computational Intelligence, 2015 IEEE Symposium Series on*. IEEE. 2015, pp. 159–166.

[9] Ofer Dekel et al. "Optimal distributed online prediction using mini-batches". In: *Journal of Machine Learning Research* 13.Jan (2012), pp. 165–202.

[10] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[11] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[12] Yehuda Koren. "The bellkor solution to the netflix grand prize". In: (2009).

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[14] Mu Li et al. "Efficient mini-batch training for stochastic optimization". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 661–670.

[15] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[16] Yurii Nesterov. "A method of solving a convex programming problem with convergence rate O(1/sqr(k))". In: *Soviet Mathematics Doklady* 27 (1983), pp. 372–376.

[17] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM Journal on Control and Optimization* 30.4 (1992), pp. 838–855.

[18] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[19] Benjamin Recht et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent". In: *Advances in neural information processing systems*. 2011, pp. 693–701.

[20] Xin Rong. "word2vec parameter learning explained". In: *arXiv preprint arXiv:1411.2738* (2014).

[21] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. "A general framework for parallel distributed processing". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1 (1986), pp. 45–76.

[22] Shai Shalev-Shwartz et al. "Pegasos: Primal estimated sub-gradient solver for svm". In: *Mathematical programming* 127.1 (2011), pp. 3–30.

[23] David Silver, Aja Huang, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[24] David Silver, Julian Schrittwieser, et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), p. 354.

[25] Ryan Tibshirani. *EECS 10-725: Optimization, Lecture 6*. Sept. 2013.

[26] Wei Xu. "Towards optimal one pass large scale learning with averaged stochastic gradient descent". In: *arXiv preprint arXiv:1107.2490* (2011).

[27] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[28] Tong Zhang. "Solving large scale linear prediction problems using stochastic gradient descent algorithms". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 116.

[29] Martin Zinkevich et al. "Parallelized stochastic gradient descent". In: *Advances in neural information processing systems*. 2010, pp. 2595–2603.